IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	KNUTSON et al.	Confirmation:	3761
Serial No.:	10/590,893	Art Unit:	2614
Filed:	August 28, 2006	Examiner:	Jamal, Alexander
For:	Acoustic Echo Canceller with Multimedia Training Signal		

DECLARATION UNDER 37 C.F.R. §1.132

I, Jens Cahnbley, declare and say:

1. That I am a citizen of Germany, and I reside at West Windsor, New Jersey 08550, United States of America;

2. That, after high school, I attended das Institut fur berufliche Qualifizierung (Institute for Professional Qualification) in Germany between 1988-1989 and, upon graduation, I was then immediately employed by Thomson SA in Germany and am now employed as a software engineer with Thomson in the United States. I have no further formal education. I was a software engineer with Thomson at the time of the Knutson invention and consider myself to be a person of ordinary skill in the subject matter of operating systems and processor load;

3. That, since 1990, I have worked in the consumer electronics industry employed by Thomson S. A. ("Thomson");

4. That I have read the above-identified patent Application, Serial No. 10/590,893 ("Application"), published as US 2007/0189508 ("Knutson");

5. That I have reviewed the Examiner's rejections of claims 7-9 and 27 for alleged non-enablement and the reasons therefor in the non-final Office Action mailed October 14, 2009 ("Non-Final Office Action");

6. That I consider myself to be a person with ordinary skill in the subject matter disclosed by Knutson as of March 5, 2004, especially as per Figure 6 and claims 7-9 and 27 where in Figure 6 there is a depiction and a subsequent discussion of whether a processor load is

high or low at box 610 and a discussion of "minimizing processor requirements during training" at box 635 and related discussion in the specification;

Average processor load

7. That, as of March 5, 2004 and without undue experimentation, it was obvious and well-known in the art that there were numerous methods available to measure average processing load. I have reviewed the Rule 132 Affidavit of my colleague, Benyuan Zhang, and cannot disagree with his comments. I offer two documents attached hereto as further evidence indicative of the knowledge of one of ordinary skill in the art as of March 5, 2004. The first document is dated 1998, comprises nine pages, and is entitled "Performance Monitor Counters" authored by Mark T. Edmead and Paul Hinsberg (hereinafter, "Edmead"). The second article is entitled "How to get CPU usage by performance counters (without PDH)" by Dudi Avramov posted 22 Dec 2002 and updated 10 Feb 2005 (hereinafter, "Avramov"). PDH stands for Performance Data Helper and PDH.dll in particular. These are API's for collecting performance data on the Windows operating system, but my personal experience is in using performance monitor counters for assessing processor load. I will now briefly discuss the relevance of each article and then address my personal experience;

8. That, as of March 5, 2004 and without undue experimentation, it was obvious from the "Processor Performance Counters" section of Edmead that a number of parameters are readily available to determine processor performance. Processor: % Processor Time, for example, is a measurement of how often the system is doing nothing subtracted from 100%. In the context of Knutson, when the processor is doing nothing is a good time for acoustic echo canceller training. Consequently, at least this parameter relates to box 610 and box 635 and would be well known to one of ordinary skill to determine and set as an average over time, depending on the processor or number and quality of system processors, for "minimizing processor requirements during training." For example, some computers have more than one processor. So System: % Total Processor Time on page 3/9 may be used as a parameter for a multi-processor system. One may average either value over time to determine an average processor load. One of ordinary skill may also set a threshold value of average processor load. A threshold is commonly used to

determine when to load the processor or processor system with tasks and when not to load the processor with tasks, in this instance, echo canceller training tasks. Processor: Interrupts/sec. is an indication of high processor load. Similarly, Processor: % Interrupt Time can be an indication of a high load. The Examiner at Page 6 (although unrelated to the rejected claims) mentions "or even every process handled by the processor." While this comment is irrelevant to the rejected claims, at page 3/9, Edmead discusses Process: Process ID whereby a Process IS counter may provide information about an identified process through API calls. One or a combination of the parameters would be used by one of ordinary skill in the art to determine high versus low processor load, to determine an average load over time on a processor or to determine a threshold for triggering when to use the processor for training an acoustic echo canceller;

9. That, as of March 5, 2004 and without undue experimentation, it is clear that Avramov reinforces Edmead. In the Introduction, Avramov provides specific guidance for a CPU usage counter. In the Comment section of Avramov, see the recommendation to use % Total processor time, a parameter suggested by Elmead. Avramov identifies in his Comment specific indices for each of several operating systems of the day. I used performance counters multiple times during my career at Thomson for determining processor load and certainly before March 5, 2004;

10. That, as of March 5, 2004 and without undue experimentation, the undersigned declares that Windows comes with a tool called PerfMon.exe (Performance Monitor) that shows the performance counters without the need to use a programming interface which may be used to determine processor load. I personally used PerfMon on many occasions for the exact purpose of checking the processor load;

11. That, as of March 5, 2004 and without undue experimentation, one of ordinary skill in the art would understand in claim 7 the concept and how to implement "minimizing use of the processor when a current load of the processor is above an average load threshold for the processor" and in claim 8/7 how "minimizing comprises collecting audio data samples from one of the microphone and the speaker and restricting use of the adaptive filter until the current load of the processor is below the average load threshold for the processor." Setting an average load threshold could be as simple as setting the load to be no load and then training. It could mean

setting a % Total processor time threshold at a value and then a little higher or higher still until the system operates as a balanced system. The examples provided are for a personal computer and WINDOWS operating system, but other operating systems have performance counters as well that may be used to practice claims 7-9 and 27. The application (real-time or non real-time) and the processor system as a whole may cause one to increase or decrease an average load threshold. As for claim 27, "operating said non-training audio application for training the acoustic echo canceller so long as a processing load on said processor of said electronic device is less than an average load of said processor of said electronic device" (my emphasis added) would be implementable by one of ordinary skill in the software engineering arts as well for similar reasons;

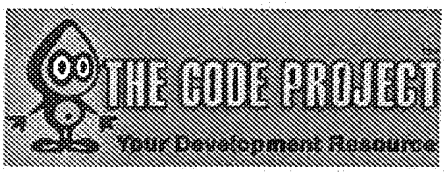
I further declare that all statements made herein of his own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like are punishable by fine or imprisonment or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application cited above or any patents issuing thereon.

Jan. 04, 2010

Jens Cahnbley

Attachments: Edmead et al., "Performance Monitor Counters," 1998

Avramov, "How to get CPU usage by performance counters (without PDH)," 2002-2005



General Reading » Hardware & System » General
License: The Code Project Open License (CPOL)
VC6Win2K, WinXP, Dev

Posted: **22 Dec 2002**
Updated: **10 Feb 2005**
Views: **300,709**
Bookmarked: **95 times**

How to get CPU usage by performance counters (without PDH)

By Dudi Avramov

Get CPU usage by performance counters without using PDH.dll.

Download source project - 7.15 Kb
Download demo project - 25.7 Kb

Introduction

The information in this article applies to Windows NT, Win2K/XP. There is no specific Win32 API that retrieves the CPU usage. An undocumented API, `NtQuerySystemInformation` in *ntdll.dll*, would help us retrieve the CPU usage. However, CPU usage can be retrieved by using performance counters. Since *PDH.dll* (Performance Data Helper) is not distributed with the Visual Studio, and not everyone has this file, I decided to do it without the help of *PDH.dll*.

The CPU usage counter is of type `PERF_100NSEC_TIMER_INV` which has the following calculation:

```
100*(1-(X1-X0)/(Y1-Y0))
  X - CounterData
  Y - 100NsTime
  Time base - 100Ns
```

where the denominator (Y) represents the total elapsed time of the sample interval and the numerator (X) represents the time during the interval when the monitored components were inactive.

My `CCpuUsage` class has a method called `GetCpuUsage` which runs through the performance objects and counters and retrieves the CPU usage. Since the CPU usage can be determined by two samplings, the first call to `GetCpuUsage()` returns 0, and all calls thereafter returns the CPU usage.

Comment

On Windows NT, CPU usage counter is '% Total processor time' whose index is 240 under 'System' object whose index is 2. However, in Win2K/XP, Microsoft moved that counter to '% processor time' whose index is 6 under '_Total' instance of 'Processor' object whose index is 238. Read 'INFO: Percent Total Performance Counter Changes on Windows 2000' (Q259390) in MSDN.

There is no difference between WinNT and Win2K/XP in the performance counters for getting CPU usage for a specific process. The counter '% processor time' whose index is 6 under the object 'Process' whose index is 230.

The Sample

```
#include "CpuUsage.h"

int main(int argc, char* argv[])
{
    int processID=0;
    CCpuUsage usageA;
    CCpuUsage usageB;
    CCpuUsage usageC;

    printf("SystemWide Cpu Usage      "
           "Explorer cpu usage        "
           "Cpu Usage for processID 0\n");
    printf("====================      "
           "====================      "
           "==========================\n");
    while (true)
    {
        // Display the system-wide cpu usage and the "Explorer" cpu usage

        int SystemWideCpuUsage = usageA.GetCpuUsage();
        int ProcessCpuUsageByName = usageB.GetCpuUsage("explorer");
        int ProcessCpuUsageByID = usageC.GetCpuUsage(processID);
        printf("%19d%%%22d%%%31d%%\r",SystemWideCpuUsage,
               ProcessCpuUsageByName, ProcessCpuUsageByID);

        Sleep(1000);
    }
    return 0;
}
```

Updates

- **6-May-2003** - Fixed bug when looking for an instance. The bug occurred because the instances in the performance counters are in Unicode. Added a sample function to get CPU usage for a specific process.
- **12-Jun-2003** - Fixed bug in `realloc` method.
- **22-Jun-2003** - Defined structure alignment to be 8 (by using `#pragma` directive). Other sizes cause the function to return 100% CPU usage. (Thanks to Scolver tip.)
- **12-Feb-2004** - Enabled performance counters of *perfOS.dll* (which holds processor counters) automatically for Win2K/XP.
- **20-May-2004** - Retrieving only the specified counter data (instead of retrieving all counters and iterating till finding the specified object) (for more information, refer to OscarTV comments).
- **03-Jun-2004** - Enabled performance counters of *perfProc.dll* (which holds process counters) automatically for Win2K/XP.
- **06-Jun-2004** - Wrapped the code into a class in order to enable getting CPU usage for different processes at the same iteration.
- **18-Nov-2004** - Extended variable from 8 bytes to 32 bytes.
- **03-Feb-2005** - Enabled getting performance counters by process ID.
- **09-Feb-2005** - Returns 0 when the specified process-ID doesn't exist.

More of my articles using performance counters

- How to get handle to any running process by its name.

License

About the Author

Dudi Avramov

Occupation: Web Developer
Location: Israel

Member

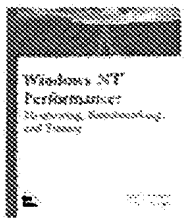
Discussions and Feedback

211 messages have been posted for this article. Visit **http://www.codeproject.com/KB/system/cpuusageByDudiAvramov.aspx** to post and view comments on this article, or click **here** to get a print view with messages.

Performance Monitor Counters

Archived content. No warranty is made as to technical accuracy. Content may contain URLs that were valid when originally published, but now link to sites or pages that no longer exist.

By Mark T. Edmead, Paul Hinsberg



Appendix A from *Windows NT Performance: Monitoring, Benchmarking, and Tuning*, published by New Riders Publishing

This appendix covers the following topics:

Processor Performance Counters. This section describes the most significant counters associated with the Processor object.

Memory Performance Counters. This section describes the most significant counters associated with the Memory object.

Disk Performance Counters. This section describes the most significant counters associated with the PhysicalDisk and LogicalDisk objects.

Network Performance Counters. This section describes the most significant counters associated with the network interface, network segment, and TCP/IP components.

This section lists all of the objects and counters that were referenced throughout the text, and some extras that will be of particular interest. There are hundreds of Performance Monitor counters that can be used for a multitude of situations. You might consider this the starter list of ones that you should be using on a regular basis. Explanations and usage of the counters is offered where appropriate. Each listing will have the following format:

[Object] : [Counter Name]. [Explanation/Usage]

Recall that an object refers to a Performance Monitor object. For each object there will be multiple counters—[Counter Name]—and potentially multiple instances. This is the case with multiple processor systems or systems with more than one hard drive. Some of the objects and counters are not present unless an NT component or other software is installed on the system. This is specifically stated in the [Explanation/Usage] section of each object/counter pairing.

On This Page

Processor Performance Counters

The Processor object is focused primarily on the CPU of the system. Note that some system have multiple processors, which will display as independent instances for each of these counters.

The counters listed in this section are all used to determine processor performance or influence other components are enforcing over the processor.

Processor : % Processor Time. This counter provides a measure of how much time the processor actually spends working on productive threads and how often it was busy servicing requests. This counter actually provides a measurement of how often the system is doing nothing subtracted from 100%. This is a simpler calculation for the processor to make. The processor can never be sitting idle waiting to the next task, unlike our cashier. The CPU must always have something to do. It's like when you turn on the computer, the CPU is a piece of wire that electric current is always running through, thus it must always be doing something. NT give the CPU something to do when there is nothing else waiting in the queue. This is called the idle thread. The system can easily measure how often the idle thread is running as opposed to having to tally the run time of each of the other process threads. Then , the counter simply subtracts the percentage from 100%.

Processor : Interrupts /sec. The numbers of interrupts the processor was asked to respond to. Interrupts are generated from hardware components like hard disk controller adapters and network interface cards. A sustained value over 1000 is usually an indication of a problem. Problems would include a poorly configured drivers, errors in drivers, excessive utilization of a device (like a NIC on an IIS server), or hardware failure. Compare this value with the System : Systems Calls/sec. If the Interrupts/sec is much larger over a sustained period, you probably have a hardware issue.

Processor : % Interrupt Time. This is the percentage of time that the processor is spending on handling Interrupts. Generally, if this value exceeds 50% of the processor time you may have a hardware issue. Some components on the computer can force this issue and not really be a problem. For example a programmable I/O card like an old disk controller card, can take up to 40% of the CPU time. A NIC on a busy IIS server can likewise generate a large percentage of processor activity.

Processor : % User Time. The value of this counter helps to determine the kind of processing that is affecting the system. Of course the resulting value is the total amount of non-idle time that was spent on User mode operations. This generally means application code.

Processor : %Privilege Time. This is the amount of time the processor was busy with Kernel mode operations. If the processor is very busy and this mode is high, it is usually an indication of some type of NT service having difficulty, although user mode programs can make calls to the Kernel mode NT components to occasionally cause this type of performance issue.

Processor: %DPC Time. Much like the other values, this counter shows the amount of time that the processor spends servicing DPC requests. DPC requests are more often than not associated with the network interface.

Process : % Processor Time. This counter is a natural choice that will give use the amount of time that this particular process spends using the processor resource. There are also % Privilege Time and % User Time counters for this object that will help to identify what the program is spending most of its time doing.

System : Processor Queue Length. Oddly enough, this processor counter shows up under the System object, but not without good reason. There is only 1 queue for tasks that need to go to the processor, even if there is more than one CPU. Thus, counter provides a measure of the instantaneous size of the queue for all processors at the moment that the measurement

was taken. The resulting value is a measure of how many threads are in the Ready state waiting to be processed. When dealing with queues, if the value exceeds 2 for a sustained period, you are definitely having a problem with the resource in question.

System : System Calls/sec. This counter is a measure of the number of calls made to the system components, Kernel mode services. This is a measure of how busy the system is taking care of applications and services—software stuff. When compared to the Interrupts/Sec it will give you an indication of whether processor issues are hardware or software related. See Processor : Interrupts/Sec for more information.

System : % Total Processor Time. This counter groups the activity of all the processors together to report the total performance of the entire system. On a single processor machine, this value will equal the %Processor Time value of the processor object.

System : % Total User Time. This is the total user time of all the processors on the system. See Processor : % User Time for more details.

System : % Total Priviedge Time. This is the total priviedge time for all processors on the system collectively. See Processor : % Priviedge Time for more details.

System : % Total Interrupt Time. This is the collective amount of time that all of the processors are spending on handling interrupts. See Processor : % Interrupt Time for more details.

Thread Object : % Processor Time. This counter takes the analysis to the next level. Typically, this counter would be for programmers, but occasionally there is a more global use for it. For example, if you are trying to examine the actions of a 16-bit process. The 16-bit application will actually be running as a thread in the NTVDM process. If you wish to see the processor usage by the 16-bit without obscuring it with the processing of the NTVDM and WOWEXEC.exe, you will want to examine the individual thread. BackOffice applications tend to have very distinct multiple threads that sometimes are worth examining individually as opposed to in a group. Often the threads of more sophisticated applications can be configured independently from the entire process.

Thread Object : ID Thread. When a process creates a thread, the system assigns a Thread ID so that it may distinguish the thread among the other threads on the system. Thread IDs are reassigned upon creation and deletion of the threads. You can not expect a thread to have the same ID each time it is created. It is important to use the Thread ID whenever you are looking at any other counters that are specific to the thread. If the thread is deleted, the performance monitor will spike indicating the thread has in fact expired.

Thread Object : Priority Base. The thread gets a base priority from the Process that created it. The priority of the thread can be adjusted by the system or through a program. This priority is used to judge when the thread is going to have access to the process and how many other threads it may jump ahead of in the processor queue of ready threads.

Process : Process ID. Each process on Windows NT gets a Process ID that identifies it as a unique process on the system. You can reference the Process ID counter to gain information about the process through API calls. The Process ID is guaranteed to remain unique to the particular process during the entire time that it is running. But, the process is not guaranteed to have the same process ID each time that it is run.

Process : % Processor Time. Each process will show up as an instance when selecting this counter. This counter will break down how much processor time each process is taking on the CPU. Don't forget to exclude the Idle and the Total counts when looking at all of the instances.

Process : % User Time. This will break down the amount of user time that each process is taking out of the total amount of processor time that the processes is using.

Top Of Page

Memory Performance Counters

The following counters all have to do with the management of memory issues. In addition, there will counters that assist in the determination of whether the problem you are having is really a memory issue.

Memory : Page Faults/sec. This counter gives a general idea of how many times information being requested is not where the application (and VMM) expects it to be. The information must either be retrieved from another location in memory or from the pagefile. Recall that while a sustained value may indicate trouble here, you should be more concerned with hard page faults that represent actual reads or writes to the disk. Remember that the disk access is much slower than RAM.

Memory : Pages Input/sec. Use this counter in comparison with the Page Faults/sec counter to determine the percentage of the page faults that are hard page faults.

Thus, Pages Input/sec / Page Faults/sec = % Hard Page Faults. Sustained values surpassing 40% are generally indicative of memory shortages of some kind. While you might know at this point that there is memory shortage of some kind on the system, this is not necessarily an indication that the system is in need of an immediate memory upgrade.

Memory : Pages Output/sec. As memory becomes more in demand, you can expect to see that the amount of information being removed from memory is increasing. This may even begin to occur prior to the hard page faults becoming a problem. As memory begins to run short, the system will attempt to first start reducing the applications to their minimum working set. This means moving more information out to the pagefiles and disk. Thus, if your system is on the verge of being truly strained for memory you may begin to see this value climb. Often the first pages to be removed from memory are data pages. The code pages experience more repetitive reuse.

Memory : Pages/sec. This value is often confused with Page Faults/sec. The Pages/sec counter is a combination of Pages Input/sec and Pages Output/sec counters. Recall that Page Faults/sec is a combination of hard page faults and soft page faults. This counter, however, is a general indicator of how often the system is using the hard drive to store or retrieve memory associated data.

Memory : Page Reads/sec. This counter is probably the best indicator of a memory shortage because it indicates how often the system is reading from disk because of hard page faults. The system is always using the pagefile even if there is enough RAM to support all of the applications. Thus, some number of page reads will always be encountered. However, a sustained value over 5 Page Reads/sec is often a strong indicator of a memory shortage. You must be careful about viewing these counters to understand what they are telling you. This counter again indicates the number of reads from the disk that were done to satisfy page faults. The amount of pages read each time the system went to the disk may indeed vary. This will be a function of the application and the proximity of the data on the hard drive. Irrelevant of these facts, a sustained value of over 5 is still a strong indicator of a memory problem. Remember the importance of "sustained." System operations often fluctuate, sometimes widely. So, just because the system has a Page Reads/sec of 24 for a couple of seconds does not mean you have a memory shortage.

Memory : Page Writes/sec. Much like the Page Reads/sec, this counter indicates how many times the disk was written to in an effort to clear unused items out of memory. Again, the

numbers of pages per read may change. Increasing values in this counter often indicate a building tension in the battle for memory resources.

Memory : Available Memory. This counter indicates the amount of memory that is left after nonpaged pool allocations, paged pool allocations, process' working sets, and the file system cache have all taken their piece. In general, NT attempts to keep this value around 4 MB. Should it drop below this for a sustained period, on the order of minutes at a time, there may be a memory shortage. Of course, you must always keep an eye out for those times when you are simply attempting to perform memory intensive tasks or large file transfers.

Memory : Nonpageable memory pool bytes. This counter provides an indication of how NT has divided up the physical memory resource. An uncontrolled increase in this value would be indicative of a memory leak in a Kernel level service or driver.

Memory : Pageable memory pool bytes. An uncontrolled increase in this counter, with the corresponding decrease in the available memory, would be indicative of a process taking more memory than it should and not giving it back.

Memory : Committed Bytes. This counter indicates the total amount of memory that has been committed for the exclusive use of any of the services or processes on Windows NT. Should this value approach the committed limit, you will be facing a memory shortage of unknown cause, but of certain severe consequence.

Process : Page Faults/sec. This is an indication of the number of page faults that occurred due to requests from this particular process. Excessive page faults from a particular process are an indication usually of bad coding practices. Either the functions and DLLs are not organized correctly, or the data set that the application is using is being called in a less than efficient manner.

Process : Pool Paged Bytes. This is the amount of memory that the process is using in the pageable memory region. This information can be paged out from physical RAM to the pagefile on the hard drive.

Process : Pool NonPaged Bytes. This is the amount of memory that the process is using that cannot be moved out to the pagefile and thus will remain in physical RAM. Most processes do not use this, however, some real-time applications may find it necessary to keep some DLLs and functions readily available in order to function at the real-time mode.

Process : Working Set. This is the current size of the memory area that the process is utilizing for code, threads, and data. The size of the working set will grow and shrink as the VMM can permit. When memory is becoming scarce the working sets of the applications will be trimmed. When memory is plentiful the working sets are allowed to grow. Larger working sets mean more code and data in memory making the overall performance of the applications increase. However, a large working set that does not shrink appropriately is usually an indication of a memory leak.

Top Of Page

Disk Performance Counters

The Disk Performance counters help you to evaluate the performance of the disk subsystem. The disk subsystem is more than the disk itself. It will include to disk controller card, the I/O bus of the system, and the disk. When measuring disk performance it is usually better to have a good baseline for performance than simply to try and evaluate the disk performance on a case by case basis.

There are two objects for the disk—PhysicalDisk and LogicalDisk. The counters for the two are identical. However, in some cases they may lead to slightly different conclusions. The PhysicalDisk object is used for the analysis of the overall disk, despite the partitions that may be on the disk. When evaluating overall disk

performance this would be the one to select. The LogicalDisk object analyzes information for a single partition. Thus the values will be isolated to activity that is particularly occurring on a single partition and not necessarily representative of the entire load that the disk is burdened with. The LogicalDisk object is useful primarily when looking at the affects or a particular application, like SQL Server, on the disk performance. Again the PhysicalDisk is primarily for looking at the performance of the entire disk subsystem. In the list that follows, the favored object is indicated with the counter. When the LogicalDisk and PhysicalDisk objects are especially different, the counter will be listed twice and the difference specifically mentioned.

PhysicalDisk : Current Disk Queue Length. This counter provides a primary measure of disk congestion. Just as the processor queue was an indication of waiting threads, the disk queue is an indication of the number of transactions that are waiting to be processed. Recall that the queue is an important measure for services that operate on a transaction basis. Just like the line at the supermarket, the queue will be representative of not only the number of transactions, but also the length and frequency of each transaction.

PhysicalDisk : % Disk Time. Much like % Processor time, this counter is a general mark of how busy the disk is. You will see many similarities between the disk and processor since they are both transaction-based services. This counter indicates a disk problem, but must be observed in conjunction with the Current Disk Queue Length counter to be truly informative. Recall also that the disk could be a bottleneck prior to the % Disk Time reaching 100%.

PhysicalDisk : Avg. Disk Queue Length. This counter is actually strongly related to the % Disk Time counter. This counter converts the %Disk Time to a decimal value and displays it. This counter will be needed in times when the disk configuration employs multiple controllers for multiple physical disks. In these cases, the overall performance of the disk I/O system, which consists of two controllers, could exceed that of an individual disk. Thus, if you were looking at the %Disk Time counter, you would only see a value of 100%, which wouldn't represent the total potential of the entire system, but only that it had reached the potential of a single disk on a single controller. The real value may be 120% which the Avg. Disk Queue Length counter would display as 1.2.

PhysicalDisk : Disk Reads/sec. This counter is used to compare to the Memory: Page Inputs/sec counter. You need to compare the two counters to determine how much of the Disk Reads are actually attributed to satisfying page faults.

LogicalDisk : Disk Reads/sec. When observing an individual application (rather a partition) this counter will be an indication of how often the applications on the partition are reading from the disk. This will provide you with a more exact measure of the contribution of the various processes on the partition that are affecting the disk.

PhysicalDisk : Disk Reads Bytes/sec. Primarily, you'll use this counter to describe the performance of disk throughput for the disk subsystem. Remember that you are generally measuring the capability of the entire disk hardware subsystem to respond to requests for information.

LogicalDisk : Disk Reads Bytes/sec. For the partition, this will be an indication of the rate that data is being transferred. This will be an indication of what type of activity the partition is experiencing. A smaller value will indicate more random reads of smaller sections.

PhysicalDisk : Avg. Disk Bytes/Read. This counter is used primarily to let you know the average number of bytes transferred per read of the disk system. This helps distinguish between random reads of the disk and the more efficient sequential file reads. A smaller value generally indicates random reads. The value for this counter can also be an indicator of file fragmentation.

PhysicalDisk : Avg. Disk sec/Read. The value for this counter is generally the number of seconds it takes to do each read. On less-complex disk subsystems involving controllers that

do not have intelligent management of the I/O, this value is a multiple of the disk's rotation per minute. This does not negate the rule that the entire system is being observed. The rotational speed of the hard drive will be the predominant factor in the value with the delays imposed by the controller card and support bus system.

PhysicalDisk: Disk Reads/sec. The value for this counter is the number of reads that the disk was able to accomplish per second. Changes in this value indicate the amount of random access to the disk. The disk is a mechanical device that is capable of only so much activity. When files are closer together, the disk is permitted to get to the files quicker than if the files are spread throughout the disk. In addition, disk fragmentation can contribute to an increased value here.

Top Of Page

Network Performance Counters

The network performance counters are not typically installed. The Network Segment object that is referred to here is installed when the Network Monitor Agent is installed. The network interface is installed when the SNMP service is installed. Many of the counters have to do with TCP/IP components, such as the SNMP service which relies on TCP/IP.

Network Interface : Bytes Sent/sec. This is how many bytes of data are sent to the NIC. This is a raw measure of throughput for the network interface. We are really measuring the information sent to the interface which is the lowest point we can measure. If you have multiple NIC, you will see multiple instances of this particular counter.

Network Interface: Bytes Received/sec. This, of course, is how many bytes you get from the NIC. This is a measure of the inbound traffic In measuring the bytes, NT isn't too particular at this level. So, no matter what the byte is, it is counted. This will include the framing bytes as opposed to just the data.

Network Interface : Bytes Total/sec. This is simply a combination of the other two counters. This will tell you overall how much information is going in and out of the interface. Typically, you can use this to get a general feel, but will want to look at the Bytes Sent/sec and the Bytes Received/sec for a more exact detail of the type of traffic.

Processor : % DPC Time. Interrupts can be handled later. These are called Deferred Procedure Calls. You will want to keep track of these as well. The combination of this time with the % Interrupt Time will give you a strong idea of how much of the precious processor time is going to servicing the network.

Processor : DPCs queued/sec. This will give you the rate at which DPC are being sent to the process queue. Unlike the Processor Queue Length and the Disk Queue Length, this value only shows you the rate at which the DPCs are being added to the queue, no how many are in the queue. Still, observing this value can give you an indication of a growing problem.

Network Segment : %Broadcasts. This value will let you know how much of the network bandwidth is dedicated to broadcast traffic. Broadcasts are network packets that have been designated as intended for all machines on the segment. Often, it is this type of traffic that has a detrimental affect on the network.

Network Segment : %Multicasts. This is a measure of the % of the network bandwidth that multicast traffic is taking. Multicast traffic is similar to the broadcast, however, there is a limited number of intended recipients. The idea was that if you can identify multiple recipients you can reduce the repetitive transmission of data. This type of transfer is used most commonly with video conferencing.

TCP : Segments Sent/sec. This is the rate at which TCP segments are sent. This is how much information that is being sent out for TCP/IP transmissions.

TCP : Segments Received/sec. Of course, the rate at which segments are received for the protocol.

TCP : Segments/sec. This is just the total of the previous two counters. This is the information being sent and received. This is a general indication of how busy the TCP/IP traffic is. The segment size is variable and thus, this does not translate easily to bytes.

TCP : Segments Retransmitted/sec. This is the rate at which retransmissions occur. Retransmissions are measured based on bytes in the data that are recognized as being transmitted before. On a Ethernet/TCP/IP network retransmissions are a fact of life. However, excessive retransmissions indicate a distinct reduction in bandwidth.

TCP : Connection Failures. This is the raw number of TCP connections that have failed since the server was started. A failure usually indicates a loss of data somewhere in the process. Data lose can occur at many locations. This could be an indication of another device being down, or problems with the client-side configuration of the software.

TCP : Connections Reset. This is typically a result of a timeout as opposed to an erroneous set of information. The reset results from the a lack of any information over a period of time.

TCP : Connections Established. This counter represents them number of connections. Unlike the other two this is more and instantaneous counter of how many TCP connections are currentiy on the system as opposed to a count of the number of successful connections.

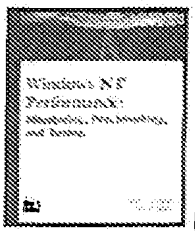
Top Of Page

About the Authors

Mark T. Edmead is president of MTE Software, Inc., a San Diego Microsoft Solutions Provider specializing in Windows NT BackOffice consulting.

Paul Hinsberg, MBA, MCSE, is the owner and operator of CRDS Inc., a computer consulting company in the Silicon Valley region.

Top Of Page



[http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=1562059424&p=technet&s=29736]
Click to order [http://www1.fatbrain.com/asp/bookinfo/bookinfo.asp?theisbn=1562059424&p=technet&s=29736]

Top Of Page